D.2 Features of OOP

Students should be able to describe the features of OOP that distinguish it from other approaches to computer programming.

Topic	Class Notes	Organised Notes
D.2.1 Define the term encapsulation.	• Encapsulation is the hiding of the internal details of a software component. The scope of data should be confined to the object which stores that data.	• **Encapsulation** is putting private variables and public methods together in a class. The variables can only be accessed from other classes by using the get and set methods. *IB Question:* Outline how security can be enhanced by using encapsulation.
D.2.2 Define the term inheritance.	• Inheritance - when a class is based on another class it can inherit its data and actions. We say that the subclass extends the superclass. • Extends - keyword creates a subclass (is a relationship) • e.g. Boss gets all the stuff an Employee has • Inheritance so we don't have to type it again	• **Inheritance** is when a class is based on another class and inherits its data and actions. The subclass extends the superclass. • Extends is a keyword that creates a subclass
D.2.3 Define the term polymorphism.	• A variable is polymorphic if it can take on an instance of a number of different (sub)classes. The different subclasses can have different implementations of methods. • Doesn't need to worry about if...	• **Polymorphism** is the ability to create a variable, a function, or an object that has more than one form. • **Polymorphism** allows you to create a subclass of an existing class and redefine how a method works, so that when the method is called by other code (or even the super class's code) your method is executed instead.
D.2.4 Explain the advantages of encapsulation.	• The fields of a class can be made read-only or write-only. • A class can have total control over what is stored in its fields. • The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code. From https://www.tutorialspoint.com/java/java_encapsulation.htm • Advantages of private variables: Only allows access to methods that access it (other classes don't have access) The variables can't be changed from other classes • Encapsulation limits dependencies in code, limits can limit errors and side-effects of changes.	Advantages of encapsulation • The variables of a class can be made read-only or write-only • A class can have total control over what is stored in variables and not have conflicting classes that conflict the variables • A class can change the data type of a field when parsing from one variable to another. The code in the class does not have to be changed. Advantages of private variables • Private variables only allows access to specific methods. Other classes that may conflict with it can't access it. • The variables can't be changed from other classes.
D.2.5 Explain the advantages of inheritance.	• Key benefit inheritance minimize the amount of dupe code in application sharing common codee amongst several subclasses. • More flexibile to change because classes inherit from a common superclass can be use interchangeable. • Reusability • Inheritance allows for similar classes to reuse the same methods without retyping the same code.	Advantages of inheritance • Inheritance minimizes the amount of duplicate code by sharing the same code with several subclasses • Inheritance makes the code more flexible because multiple classes can inherit from a common superclass and the code can be used wherever. • The same code can be reused without having to rewrite it.
D.2.6 Explain the advantages of polymorphism.	• Polymorphism allows external code to call the methods of the superclass without needing to know the implementation details of the method.	Advantages of polymorphism: - Polymorphism allows external code to call the methods of the superclass without needing to know the implementation details of a method - Allows methods that perform similar or closely related functions to be accessed through a common name
D.2.7 Describe the advantages of libraries of objects.	• Provides easy method of reusing objects • Saves time by copy/pasting already made code to the program • Restricts certain methods and actions if you're not using them • Examples of java libraries Java.lang Java.lang.Math Java.math Java.io Java.awt Many more...	Advantages of libraries of objects • Using libraries provide an easy method of reusing objects. • It saves time by copy and pasting already made code to the program • Libraries lets you dictate what objects you need and objects you don't need. *IB Question:* Explain why it is convenient to store data as objects. [3 marks] - It is safer; - Because using different arrays would make keeping data for one object more difficult to keep together - Less chance of data being overwritten from other modules - It is more convenient for programming - As programming with objects allows advantages of modularity - e.g. use of a team of programmers takign specific modules
D.2.8 Describe the disadvantages of OOP	• It can be difficult to initially comprehend concepts such as inheritance and polymorphism	Disadvantages of OOP: - It can be difficult to initially comprehend concepts such as inheritance and polymorphism. - It can be difficult and unnecessary to use objects to solve particular problems

D.3 Program development

7:32 PM

Topic	Class Notes	Organised Notes
D.3.1 Define the terms: class, identifier, primitive, instance variable, parameter variable, local variable.	• Class - a template for objects Anything called .java is a class Class contains methods and variables (things about an object and things an object can do.) • Without an object, class can't be used • Can have multiple objects in class but cannot have the same identifier • Class is a template that outlines things about an object and thing that object can do • .java class (within that, you create instance of object) • Open - contains variable called application • Error - null pointer exception Null means there is nothing there • Pointing to a reference point in memory but it isn't there • Error is runtime (won't tell there is a problem unless run) • Compile and syntax - synonymous • Syntax error occurs when something is written incorrectly • Null - an object is not available, index number is not there in array • Logical error - everything is right but it doesn't do what is expected. • Another logical error is dividing by 0 • Identifier • Label for a variable, object, method, class • Anything that has a name that can be referenced has an identifier • Identifier are ways in which something can be referenced (class, object, variable, method, all need identifiers) • Need to be able to understand what that thing • Primitive - the simplest form of data (storing) • 8 primitive data sizes with bit sizes Boolean - 1 bit Byte - 8 bit Char - 16 bit or 7/8 bit (code in java and have ability to have multiple languages to code in) Short - 16 bit (whole number) Int - 32 bit Float - 32 bit Long - 64 bit Double 64 bit • Parameter variable - variable used within a method's parameters • Formal parameter - data type that is expected within method signature • Public void Customer (String name) { } Customer ("Hello"); • Parameter - data types and names that are used within methods	• A **class** is a template for objects which defines the attributes and behaviours of objects within the class. It contains methods and variables. • An **identifier** is a name that refers to a variable, method, class, package. • A **primitive** data type is the simplest form of data within the Java language (boolean, byte, char, short, int, float, long, double) • An **instance variable** is a variable that is declared in a class but outside a method, constructor or block • A **local variable** is a variable that is declared in methods, constructors or blocks. It can only be used within a method. • A **formal parameter** is the data type that is expected within the method signature. Declaring an array: `int [ ] num = new int [ 10 ];` type of each element name of array subscript (integer or constant expression for number of elements.)
D.3.2 Define the terms: method, accessor, mutator, constructor, signature, return value.	• Method - behaviour of object • Algorithm that is called upon an object which causes an object to do something • Methods are invoked on an object. • Constructor • Accessor - method that retrieves a variable's value (e.g. getAge()) • Mutator - sets the value (e.g. setAge()) • Signature Method signature is method name and formal parameters combined • Return value Value that is returned after a method is called Uses key word return	• A **method** is an algorithm that is invoked on an object which causes an object to do something • A **constructor** method is method that is invoked when creating a new instance of an object. A constructor method has the exact same name as the class it's in. • An **accessor** is a method that retrieves a variable's value • A **mutator** is a method that sets a variable's value • A **signature** is a method name and formal parameters combined • A **return value** is a value that is returned after a method is called • **Validation** is checking if two values match • **Verification** is checking a value with another value that is already stored Accessor: `public String getFirstName() { return firstName; }` Mutator:

```
public boolean isEmpty() {
    return (numItems == 0);
}
```

isFull()
```
public boolean isFull() {
    if (numItems == MAX_LIST) { // checks if the number of items is at the maximum capacity
        return true;
    } else {
        return false;
    }
}
```

peek()
```
public int peek() {
    return(items[0]);
}
```

Stack:

push()
```
public void push (int item) {
    list.addHead(item);
}
```

pop()
```
public int pop() {
    int item = peek();
    list.delete(0);
    return item;
}
```

peek()
```
public int peek() {
    return list.peek();
}
```

isEmpty()
```
public boolean isEmpty() {
    return list.isEmpty();
}
```


Queue:

enqueue()
```
public void enqueue (int item) {
    list.addHead(item);
}
```

dequeue()
```
public void dequeue() {
    list.delete(this.size());
}
```

isEmpty()
```
public boolean isEmpty() {
    return list.isEmpty();
}
```